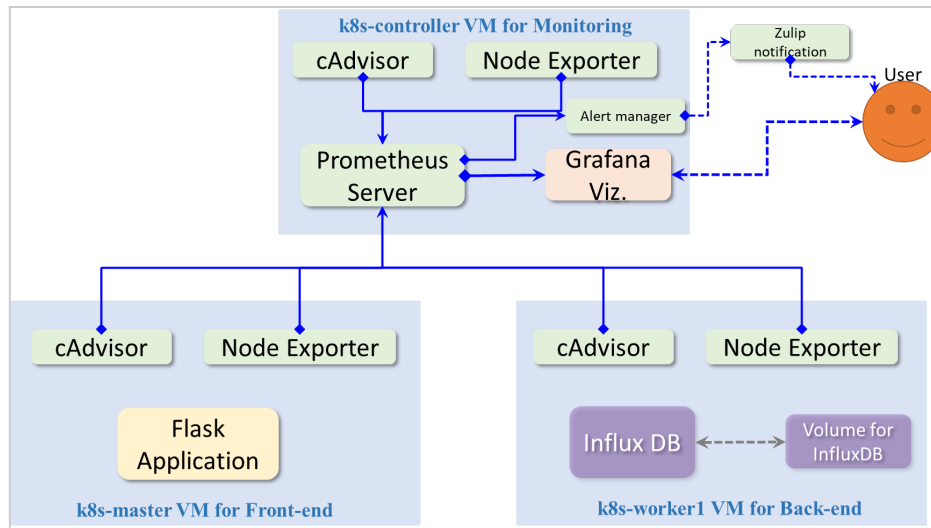


Practice Session-09 : Monitoring with Prometheus

Make sure that you have already gone through [Lab-08](#).



Prerequisite

- Basic knowledge on Python, pandas library, csv file format
- Basic knowledge on YAML
- Familiar with Dockerfile
- Familiar with GitLab webIDE or Git commands

1. Environment Setup

Step 1.1. Create a Gitlab Project

1. Create a blank project in the gitlab with name `lab09-monitoring-prometheus` under your group
`Devops2022fall/students/devops2022Fall-<lastname>-<studyCode>`
2. In the following exercises and steps, you may use **WebIDE** to add, remove, edit the files and folder in your Gitlab project.

Step 1.2. VMs Configuration

1. By now, you have four VMs (as below). We will reuse these VMs in this session.
 - a. k8s-controller
 - b. k8s-master
 - c. k8s-worker1

2. Install and register gitlab-runners with below configuration:
 - a. Gitlab-runner in **k8s-controller** VM should have **controller** tag, and **shell** executor
 - b. Gitlab-runner in **k8s-master** VM should have **master** tag, and **shell** executor
 - c. Gitlab-runner in **k8s-worker1** VM should have **worker1** tag, and **shell** executor

Step 1.3. Notes

Below some of the instructions are descriptive. That means you may get errors while executing the given minimal version of the commands. You need to investigate and fix those errors. For this, you may need to google and debug the error by yourself.

Remember that, we may see the commit history and the pipelines while grading your submission.

2. Setting up Prometheus and associated services on k8s-controller VM

In this exercise, we are going to set up the following services in the **k8s-controller** VM. We will install all in separate containers.

| Service Name | Purpose | Port numbers | For more information |
|---------------|---|--------------|---|
| Prometheus | It is an open-source monitoring solution that pulls the performance metrics from the remote VMs and containers based on specific time intervals. Further, trigger the alerts based on PromoQL queries | 9090 | https://prometheus.io/ |
| Alertmanager | Send notification to web hook endpoints based on trigger from prometheus events | 9093 | https://prometheus.io/docs/alerting/latest/alertmanager/ |
| cadvisor | Pull docker container metrics such as CPU, Memory, Disk and Network utilization metrics. | 8081 | https://github.com/google/cadvisor |
| node-exporter | Pull the VM metrics such as CPU, Memory, Disk and Network utilization metrics. | 9100 | https://prometheus.io/docs/guides/node-exporter/ |
| grafana | Visualization of metrics from Prometheus using nice interactive dashboards | 3000 | http://grafana.com |

In the following steps, we assume that you are using **WebIDE** to edit your GitLab project.

Imp: It is assumed that the **above port numbers** are available on k8s-controller, k8s-master, and k8s-worker1 VMs. If you get any error in later steps, you may choose another port number. Just make sure that it is consistent across the whole deployment.

Step 2.1. Setting up of services

1. Create a directory **monitor** inside your project's root directory and write a shell script **monitor/install_services.sh** to install prometheus, node-exporter, cadvisor, and grafana using docker commands as shown below.

Filename: `/monitor/install_services.sh`

```
docker stop $PROMO_CONTAINER || true
docker rm $PROMO_CONTAINER || true

docker stop $CADVISOR_CONTAINER || true
docker rm $CADVISOR_CONTAINER || true

docker stop $NODE_EXPORTER_CONTAINER || true
docker rm $NODE_EXPORTER_CONTAINER || true

docker stop $GRAFANA_CONTAINER || true
docker rm $GRAFANA_CONTAINER || true

# CADVISOR container for controller VM
# UPDATE THE BELOW DOCKER STATEMENT IF REQUIRED
docker run \
  --volume=/:/rootfs:ro \
  --volume=/var/run:/var/run:ro \
  --volume=/sys:/sys:ro \
  --volume=/var/lib/docker:/var/lib/docker:ro \
  --volume=/dev/disk:/dev/disk:ro \
  --publish=8081:8080 \
  --detach=true \
  --name=$CADVISOR_CONTAINER \
  --privileged \
  --device=/dev/kmsg \
  gcr.io/cadvisor/cadvisor:v0.45.0

# NODE-EXPORTER container for controller VM
# UPDATE THE BELOW DOCKER STATEMENT IF REQUIRED
docker run -d -p 9100:9100 --name $NODE_EXPORTER_CONTAINER
prom/node-exporter

# Grafana for visualization of resources used by containers and VMs
# UPDATE THE BELOW DOCKER STATEMENT IF REQUIRED
docker run -d -p 3000:3000 --name $GRAFANA_CONTAINER grafana/grafana:6.5.0

# PROMETHEUS for collecting metrics using NODE-EXPORTER and CADVISOR
# UPDATE THE BELOW STATEMENTS IF REQUIRED
rm -f $HOME/monitor/prometheus.yml || true
rm -r $HOME/monitor || true
mkdir -p $HOME/monitor/
cp $PWD/monitor/prometheus.yml $HOME/monitor/
docker run -d -p 9090:9090 \
  -v $HOME/monitor/prometheus.yml:/etc/prometheus/prometheus.yml \
  --name $PROMO_CONTAINER \
  prom/prometheus
```

2. Create a prometheus configuration file **monitor/prometheus.yml** to add the targets **node_exporter** and **cadvisor**. The below code is just for reference. The k8s-master, k8s-worker1 node IPs can be added as marked below and their services (node-exporter, cadvisor) are going to be configured in later exercises.

Filename: /monitor/prometheus.yml

```
global:
  scrape_interval: 15s
  scrape_timeout: 10s
  evaluation_interval: 15s

scrape_configs:
- job_name: prometheus
  honor_timestamps: true
  scrape_interval: 15s
  scrape_timeout: 10s
  metrics_path: /metrics
  scheme: http
  follow_redirects: true
  static_configs:
    - targets:
        - localhost:9090

- job_name: 'controller-node-exporter'
  static_configs:
    - targets: ['<k8s-controller_VM_EXT_IP>:9100']

- job_name: 'controller-cAdvisor'
  static_configs:
    - targets: ['<k8s-controller_VM_EXT_IP>:8081']

- job_name: 'master-node-exporter'
  static_configs:
    - targets: ['<k8s-master_VM_EXT_IP>:9100']

- job_name: 'master-cAdvisor'
  static_configs:
    - targets: ['<k8s-master_VM_EXT_IP>:8081']

- job_name: 'worker1-node-exporter'
  static_configs:
    - targets: ['<k8s-worker1_VM_EXT_IP>:9100']

- job_name: 'worker1-cAdvisor'
  static_configs:
    - targets: ['<k8s-worker1_VM_EXT_IP>:8081']
```

3. Create a `.gitlab-ci.yml` file inside the root directory of your project with a job to create monitoring services in the monitor VM. Here, we will define all the variables that are used in the previous shell script and `prometheus.yml` configuration file.

Filename: .gitlab-ci.yml

```
variables:
  PROMO_CONTAINER : prom_monitor
  CADVISOR_CONTAINER : cadvisor
  NODE_EXPORTER_CONTAINER : nodeExporter
  GRAFANA_CONTAINER : grafana
  ALERT_CONTAINER : alertmanager

stages:
  - monitor-stage
```

```
monitor-job:
  tags:
    - controller
  stage: monitor-stage
  script:
    - chmod +x ./monitor/install_services.sh
    - bash ./monitor/install_services.sh
  after_script:
    - sleep 60
    - docker ps -a
  only:
    changes:
      - monitor/*
```

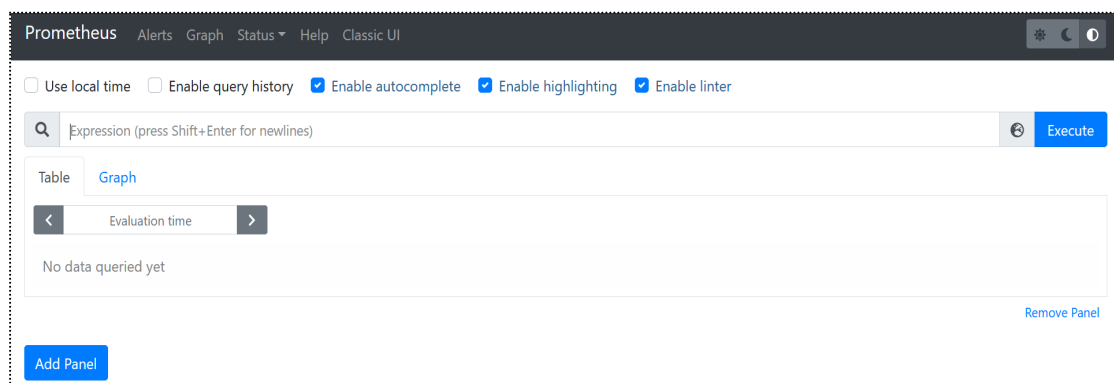
4. Commit and push the changes with message *Step 2.1 added monitor service on Controller VM*. Further, watch the jobs running in **CI-CD-->Pipelines**. After success, check for all services running in the k8s-controller VM using `docker ps` command. You may also use any web browser and visit the `/metrics` endpoints exposed by the containers (node-exporter, cadvisor) using `http://<controller_VM_EXT_IP>:<cadvisor_container_port>/metrics`

Step 2.2. Working with Prometheus dashboard and PromoQL queries

In this task, you're going to explore prometheus service for monitoring the VMs and containers metrics pulled from node-exporter and cadvisor. Prometheus provides a functional query language called **PromQL** (Prometheus Query Language) that lets the user select and aggregate time series data in real time (Ref:

<https://prometheus.io/docs/prometheus/latest/querying/basics/>)

1. Now, access the prometheus service in the browser at http://controller_VM_EXTERNAL_IP:9090 . A sample output is shown below.

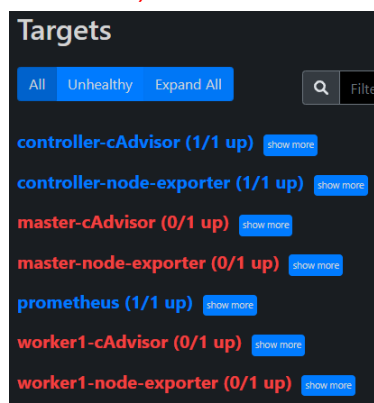


You can have a look into Prometheus dashboard to understand the different tabs.

Alerts: List of alerts defined to trigger notifications, when defined events occur such as *application container attends a down time* (This will be precisely explored further in the below Exercises)

Graph: It has a Query browser to write the PromoQL queries to get the metrics scraped from cadvisor and node exporter.

Status: This has a set of sub tabs. For example **Configuration**, that describes the configuration file `prometheus.yml` with targets. Now goto **Status-->Targets**, which will display a set of targets (endpoints) from where it pulls the resource utilization metrics. (If you see the node-exporter and cAdvisor on other VMs are down with red colour then please ignore, as these services are going to be up in later exercises.)

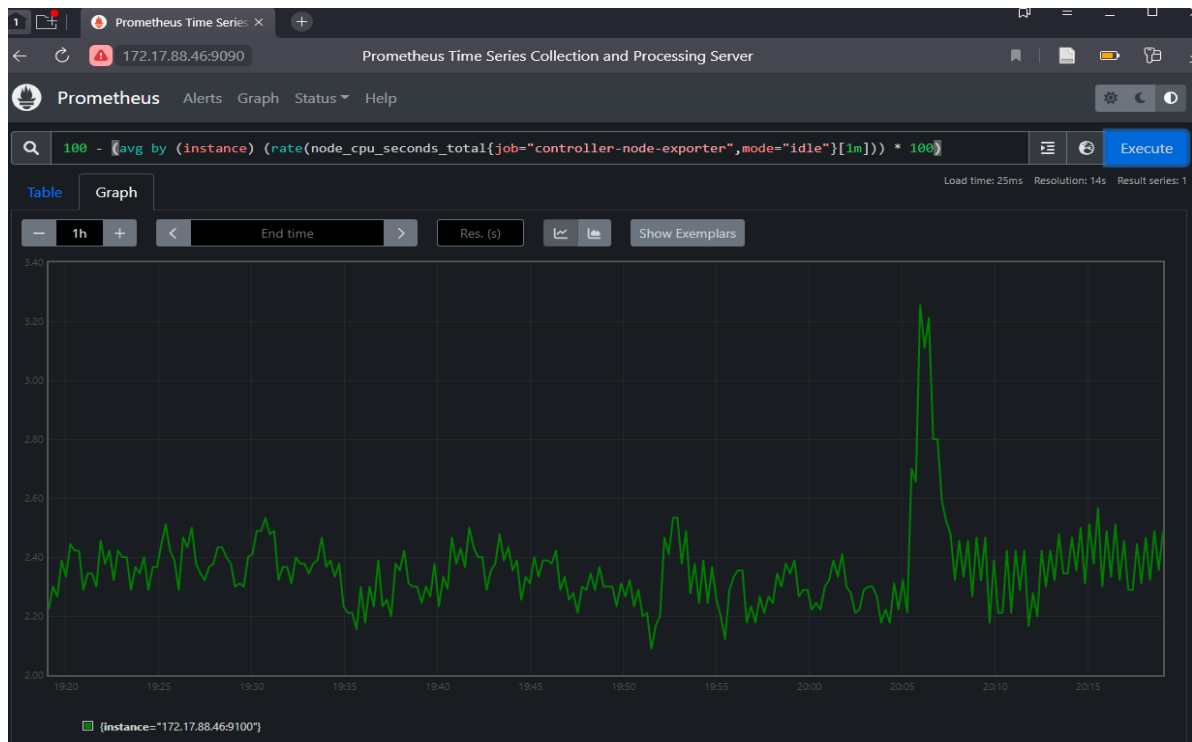


- Now, let us write PromoQL queries to see the current resource utilization of the **controller** VM.

Try the below set of queries by copy and paste into **Graph-->Expression** and then click on Execute. There are two types of outputs(Table and Graph) and see the graph windows for output.

| Metric Name | PromoQL query |
|-------------------------------------|--|
| Average CPU utilization | $100 - (\text{avg by (instance)} (\text{rate}(\text{node_cpu_seconds_total}\{\text{job}=\text{"controller-node-exporter"}, \text{mode}=\text{"idle"}\}[1\text{m}])) * 100)$ |
| Average Memory utilization | $100 * (1 - ((\text{avg_over_time}(\text{node_memory_MemFree_bytes}\{\text{job}=\text{"controller-node-exporter"}\}[1\text{m}]) + \text{avg_over_time}(\text{node_memory_Cached_bytes}\{\text{job}=\text{"controller-node-exporter"}\}[1\text{m}]) + \text{avg_over_time}(\text{node_memory_Buffers_bytes}\{\text{job}=\text{"controller-node-exporter"}\}[1\text{m}])) / \text{avg_over_time}(\text{node_memory_MemTotal_bytes}\{\text{job}=\text{"controller-node-exporter"}\}[1\text{m}]))))$ |
| Disk utilization | $(\text{avg}(\text{node_filesystem_size_bytes}\{\text{device!}=\text{"rootfs"}, \text{job}=\text{"controller-node-exporter"}\}) \text{ by (device)} - \text{avg}(\text{node_filesystem_free_bytes}\{\text{device!}=\text{"rootfs"}, \text{job}=\text{"controller-node-exporter"}\}) \text{ by (device)}) / \text{avg}(\text{node_filesystem_size_bytes}\{\text{device!}=\text{"rootfs"}, \text{job}=\text{"controller-node-exporter"}\}) \text{ by (device)})$ |
| Network Receive packets per second | $\text{rate}(\text{node_network_receive_packets_total}\{\text{job}=\text{"controller-node-exporter"}\}[1\text{m}])$ |
| Network Transmit packets per second | $\text{rate}(\text{node_network_transmit_packets_total}\{\text{job}=\text{"controller-node-exporter"}\}[1\text{m}])$ |

Sample output of first query “Average CPU utilization” is shown below:



Similarly, write a query for the network (transmit and receive measured in bytes) for 60seconds interval utilization. You can use

`node_network_receive_bytes_total` and
`node_network_transmit_bytes_total` metric.

Take the screenshots with your IP visible. Keep the screenshots in the project's root directory with the name `Network_Receive_Ex2_2.jpg` and `Network_Transmit_Ex2_2.jpg`.


Commit and push the changes with message `Step 2.2 Getting comfortable with PromoQL..`

Step 2.3. Working with Prometheus data source in Grafana.

Grafana is a visualization service that helps to create a nice dashboard and alert mechanisms for time series data. It has a set of ready-made dashboards available at <https://grafana.com/grafana/dashboards/>. Those can be imported into grafana service. Similarly we use a ready-made dashboard with prometheus as a datasource to visualize the container metrics pulled by prometheus using cAdvisor. These dashboards are written with extension `.json`.

In this task, you will learn about creating nice visualization dashboards for resource utilization of containers.

1. Go to grafana service dashboard http://controller_VM_EXTERNAL_IP:3000, here the default username: `admin` and password: `admin`.
2. Add the datasource as Prometheus in the grafana service on left side panel **Configuration-->Data sources-->Add data source-->Select Prometheus**, Under this **HTTP-->URL-->**`http://controller_VM_EXTERNAL_IP:9090` and Click on **Save and Test** button. After that click on the **Back** button.

- Let us create a dashboard for visualization of container metrics, create `/monitor/grafana_container_monitoring.json` file and copy and paste the content from here [grafana_container_monitoring.json](#) (this json file is tested ok and should work without any error).
- Now to create a dashboard, Goto grafana service mouse over to  icon **Create-->Import-->Paste the json(paste the contents of grafana_container_monitoring.json)-->Load** and save the dashboard with a name. After successful import you should have a dashboard something similar to below:



- Now focus on the queries used to get the metrics, for example mouse over to **Total CPU Usage** → **Edit** and understand the query that gets the metrics of CPU utilization in percentage from the last 5m.
- Add one more panel to display CPU Usage by *specific* container from the last 30m. (How add panel in grafana- [Adding Panel in Grafana](#))

Take the screenshot with IP visible and add to the project's root directory. Keep the screenshots in the project's root directory with the name `CPU_container_2.3.jpg`.

- Commit and push** the changes with message *Step 2.3: Got familiar with Grafana Dashboards.*

3. Setting up Flask services on k8s-master VM

In this exercise, you're going to create a flask application to read a IoT data from a backend influxdb service and display it on the webpage, Further you will monitor the resource

utilization metrics. The flask application will be deployed on k8s-master VM. In later exercise, we will deploy influxdb on k8s-worker1 VM.

Henceforth, “k8s-master VM” and “master VM” are used interchangeably and should be treated as the same.

Step 3.1 Creating a flask application

1. Create a directory `flask-frontend` inside your project's root directory
2. Reuse the `co2.csv` file from the previous sessions. Keep the file `co2.csv` file inside the `flask-frontend` directory.
3. Create a directory with name `templates` under the inside `flask-frontend` directory. `templates` directory is used to store the html files. Now create an `index.html` with a single line `<!DOCTYPE html>`.
4. Create a controller (*sample code given below*) to read the data from influxdb and update the same in `index.html` when HTTP GET request is invoked. Here, `get_influxdata()` will retrieve the data from the influxdb backend service using the python influxdb client.
5. Fix the error caused by inconsistent use of tabs and spaces in indentation. Also fix if there are any other minor errors.

Filename: `/flask-frontend/controller.py`

```
import pandas as pd
import os
import datetime
from influxdb import InfluxDBClient
from influxdb.exceptions import InfluxDBClientError, InfluxDBServerError
from requests.exceptions import RequestException
# get the current working directory
cwd = os.getcwd()

# Get the data from influxdb
def get_influxdata():
    dbname = os.getenv('INFLUX_DB_NAME')
    host = os.getenv('INFLUX_DB_HOST')
    client = InfluxDBClient(host=host, port=8086)
    # Check for data present in influxdb else display no data present
    try:
        client.switch_database(dbname)
        query = "select * from CO2;"
        df = pd.DataFrame(client.query(query).get_points())
        index = open(cwd+'/templates/index.html', "a")
        index.write("<br/>")
        index.write(df.to_html())
        index.close()

    except (RequestException, InfluxDBClientError, InfluxDBServerError) as ex:
        index = open(cwd+'/templates/index.html', "a")
        index.write("<br/>")
        index.write("<br/>")
        index.write("Exception occurred while retrieving the data from influxdb:" +
str(ex))
        index.write("Hence, data from "+dbname+" influxdb database cannot be
displayed")
        index.close()
    return "Success"
```

6. Create a flask listener with the name `app.py` with the following sample code. Fix the error caused by inconsistent use of tabs and spaces in indentation. Also fix if there are any other minor errors.

Filename: `/flask-frontend/app.py`

```
#Flask imports
from flask import Flask, render_template, send_file, make_response, url_for
from flask import Response
from controller import *

# Create a flask app
app = Flask(__name__)

# main route
@app.route('/')
def home():
    get_influxdata()
    return render_template('index.html', name='IoT data')
if __name__ == '__main__':
    app.run(debug = True, host='0.0.0.0', port=5000)
```

7. Create `/flask-frontend/requirements.txt` file to install pip packages such as flask, pandas and influxdb python client
8. Now, write a Dockerfile to build your flask application. *Sample code* shown below:

Filename: `/flask-frontend/Dockerfile`

```
FROM python:3.8-slim-buster
WORKDIR /app
ENV INFLUX_DB_NAME=<your_influxdb_name>
ENV INFLUX_DB_HOST='<worker1_VM_EXT_IP>'
COPY ./flask-frontend/requirements.txt requirements.txt
RUN pip3 install -r requirements.txt
COPY ./flask-frontend/ .
CMD [ "python3", "-m", "flask", "run", "--host=0.0.0.0"]
```

9. Write a shell script `/flask-frontend/install_services.sh` to install flask application, node-exporter, and cadvisor using docker commands as shown below.

Filename: `/flask-frontend/install_services.sh`

```
This is similar to monitor/install_services.sh and modify this script accordingly

1. Stop and remove the existing node exporter container
2. Stop and remove the existing cadvisor
3. Stop and remove the existing flask container

# CADVISOR to monitor the container resource utilization of frontend system
Write a docker command to run the cadvisor container

# NODE-EXPORTER to monitor the resource utilization of frontend system
Write a docker command to run the node-exporter container

# Build flask application container image
docker build -t $TEST_IMAGE_NAME -f ./flask-frontend/Dockerfile .

# Run the flask application container
docker run -d -p 8082:5000 --name $FLASK_CONTAINER $TEST_IMAGE_NAME
```

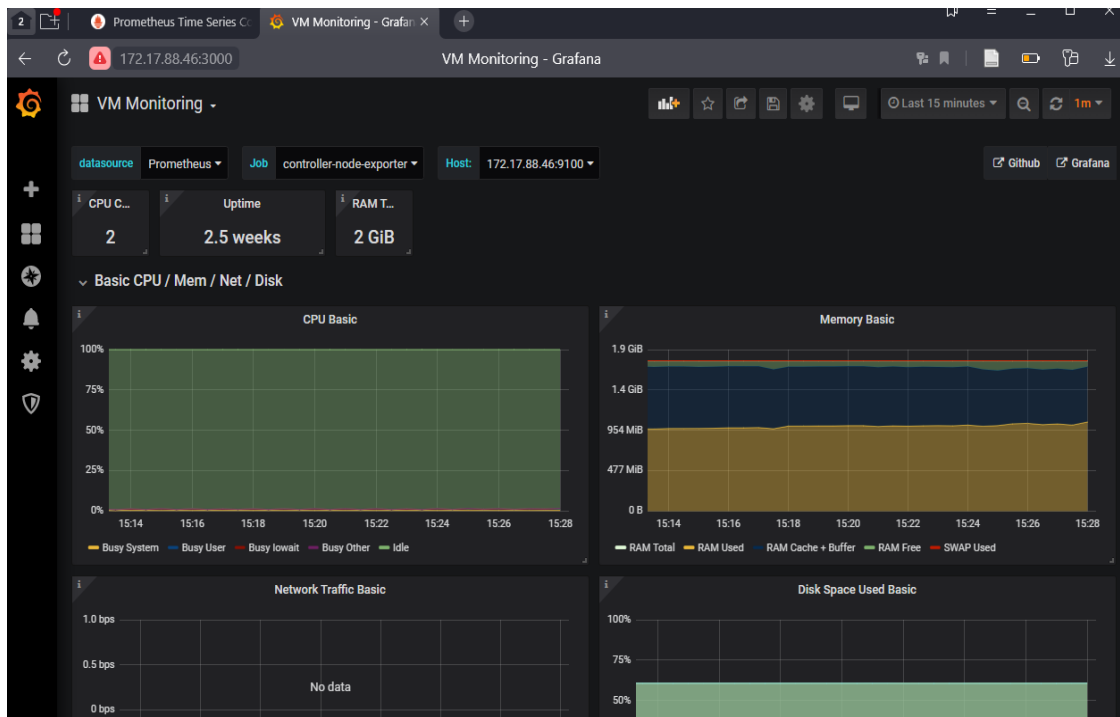
10. Now, finally update to the `.gitlab-ci.yml` file to install and set up flask-frontend services including the Flask application. This is similar to the job mentioned in previous steps.
 - a. Add the variables (you may change the values as per your convenience)
 - i. `INFLUX_DB_NAME: influxdb-co2`
 - ii. `INFLUX_DB_HOST : <worker1_VM_EXT_IP>`
 - iii. `FLASK_CONTAINER : flask_cont`
 - iv. `TEST_IMAGE_NAME: flask_image`
 - b. Add job with name `flask-frontend` and should have the following settings
 - i. Add tag as `master`
 - ii. `stage: build`
 - iii. `script: chmod +x ./flask-frontend/install_services.sh, bash ./flask-frontend/install_services.sh`
 - iv. Run only when the contents of the **flask-frontend** directory changed.
 - c. Add below `after_script` block under `flask-frontend` job

```
after_script:
- sleep 60
- docker ps -a
- docker logs $FLASK_CONTAINER
```
11. **Commit and push** the changes with message `Exercise 3: step 3.1, added frontend services on master node.`
12. Watch the completion of job in **CI/CD-->jobs** of gitlab
13. Once job is succeeded then check the running of your flask application using `http://master_VM_IP:8082`

It may display that no data found in the influxdb due to exceptions occurred while retrieving the data. This is because no backend influxdb service is running as of now.

Step 3.2. VM resource utilization in Grafana

1. Let us visualize the k8s-master VM resource metrics using grafana dashboard using ready made template [grafana_VM_monitoring.json](#). Create `./monitor/grafana_VM_monitoring.json` and copy the content of [grafana_VM_monitoring.json](#) (this json file is tested ok and should work without any error).
2. Now to create a dashboard, Goto grafana service Create-->Import-->Paste the json and save the dashboard. After successful import you should see the dashboard *similar* to below.



Take the screenshot and keep the image file with the name **grafana_VM_monitoring.jpg** in the project's root directory.

Commit and push the changes with message **Exercise 3: step 3.2, added grafana_VM_monitoring dashboard.**

4. Setting up of backend services on worker1 VM

In this exercise, you're going to set up influxdb as a backend service that stores the IoT data in time series manner. The influxdb will be installed in the **k8s-worker1** VM. For further information on influxdb can found here [influxdb](#)

Step 4.1 Installing influxdb service and associated services

1. Create a directory **influx-backend** inside your project's root directory.
2. Write a shell script **install_services.sh** (sample code given below) to install influxdb, node-exporter, and cadvisor using docker commands.

Filename: `/influx-backend/install_services.sh`

This is similar to monitor/install_services.sh and modify this script accordingly

1. Stop and remove existing node exporter container
2. Stop and remove existing cadvisor
3. Stop and remove existing influxdb container

Create volume for influxdb

```

docker volume create $INFLUX_VOL || true

# CADVISOR to monitor the container resource utilization of frontend
system
# Docker command to run the cadvisor container

# NODE-EXPORTER to monitor the resource utilization of frontend
system
# Docker command to run the node-exporter container

#INFLUXDB container
docker run -d \
  -p 8086:8086 \
  --name $BACKEND_CONTAINER \
  -v $INFLUX_VOL:/var/lib/influxdb2 \
  influxdb:1.1.1

pip install -r ./influx-backend/requirements.txt || true

# Insert CSV data in to influxdb
python3 ./influx-backend/csvToInflux.py

```

3. Create a python script `/influx-backend/csvToInflux.py` (sample code given below) to insert csv data influxdb .

Filename: `/influx-backend/csvToInflux.py`

```

import pandas as pd
from influxdb import InfluxDBClient
import sys

# get $INFLUX_DB_NAME and $CSV_FILENAME from the environment variables

client = InfluxDBClient(host=os.getenv("INFLUX_DB_HOST") , port=8086)

# Check for existing data base
client.drop_database(dbname)
client.create_database(dbname)
client.switch_database(dbname)

filename = sys.argv[2]
file_path = open(filename,'r')
csvReader = pd.read_csv(file_path)

for row_index, row in csvReader.iterrows() :
    tags = row[3]
    json_body = [{
        "measurement": row[0],
        "time": row[2],
        "tags": {
            "host": tags
        },
        "fields": {
            "unit": row[4],
            "value": row[5],
        }
    }]
    print(json_body)

```

```
client.write_points(json_body)
```

4. Create a `/influx-backend/requirements.txt` and add `pandas, influxdb` to that file.
5. Now, finally update the existing `.gitlab-ci.yml` file to install and set up backend services similar to previous steps.
 - a. Add the variables (you may change the values as per your convenience)
 - i. `CSV_FILENAME: co2.csv`
 - ii. `INFLUX_DB_NAME : <your_influxdb_name>` (should be same to the previous one)
 - iii. `INFLUX_VOL : influxdb-storage`
 - iv. `BACKEND_CONTAINER : influxdb_cont`
 - b. Add job with name `influx-backend` and should have following settings
 - i. Add tag as `worker1`
 - ii. `stage: build`
 - iii. `script: chmod +x ./influx-backend/install_services.sh, bash ./influx-backend/install_services.sh`
 - iv. Run only when the contents of the `influx-backend` directory changed.
6. **Commit and push** the changes with message `Exercise 4: step 4.1, added influx-backend services on worker1.`
7. Check your flask application using `http://master_node_VM_EXT_IP:8082`
It should now display the retrieved IoT data from influxdb.

Step 4.2. VM resource utilization in Grafana

Now, go to previously created dashboard in Step 3.2 , visualize the metrics of **k8s-worker1** VM

Take the screenshot with IP visible (**VM_monitoring_Step_4_2.jpg**). Add the screenshot to the project's root directory.

Step 4.3. Container resource utilization in Grafana

Go to grafana's "**Monitoring of Containers**" dashboard and see the updated container resource utilization dashboard.

Take the screenshot with IP visible **Monitoring_of_Containers_Step_4_3.jpg**. Add the screenshot to the project's root directory.

Commit and push the changes with message `Exercise 4: step 4.2 and 4.3: Checking VMs and containers dashboard.`

5. Deliverables: Working with Prometheus alerts

In this section, you will work to trigger the alerts to notify the users in a Zulip stream when certain events occur in the monitoring metrics, for example, CPU utilization of the container exceeds 90% then notify the user. In this experiment you will use alertmanager service along with prometheus.

As an example you will monitor the down time of the flask application created in Exercises 3 and send an alert notification to zulip topic “DevOps-alerts-topics-lab09” if the down time occurs in the last 10seconds.

Below are some hints to perform this:

Below codes are just for reference. Copy the code at your own risk. We strongly recommend NOT to simply copy paste the code. Make yourself comfortable with YAML.

Step 5.1 Create alertmanager service and associated configuration files

1. Create a file `rules.yml` inside **monitor** directory (sample code given below) with rules to monitor and trigger. In this file, you should write a PromoQL query to monitor the event
 - a. `expr:`

```
absent(container_start_time_seconds{name="<Your_flask_application_name or the flask container name>"})
```
2. This file will be referred to in the `prometheus.yml` configuration file.
3. More about alerting rules:
https://prometheus.io/docs/prometheus/latest/configuration/alerting_rules/#alerting-rules

Filename: `/monitor/rules.yml`

```
groups:
- name: AllInstances
  rules:
  - alert: InstanceDown
    # Condition for alerting
    expr: absent(container_start_time_seconds{name="flask_application_container"})
    for: 10s
    # Annotation - additional informational labels to store more information
    annotations:
      title: 'Instance {{ $labels.name }} down'
      description: '{{ $labels.name }} instance has been down for more than 10 seconds.'
    # Labels - additional labels to be attached to the alert
    labels:
      severity: 'critical'
```

4. Create a file `alertmanager.yml` inside **monitor** directory (sample code given below). Here we define the notification endpoint for an alertmanager to send notification to the end user, for example in our case it is Zulip topic “DevOps-alerts-topics-lab09”. This is used by alertmanager service to invoke the notification endpoint when alerts are triggered by prometheus service.

Filename: `/monitor/alertmanager.yml`

```

route:
  group_by: [instance, severity]
  receiver: 'Zulip-notifications'
  routes:
    - receiver: 'Zulip-notifications'
      matchers:
        - alertname="servicedown"

receivers:
- name: 'Zulip-notifications'
  webhook_configs:
    - url:
      "https://zulip.cs.ut.ee/api/v1/external/alertmanager?api_key=abcdefghi&stream=DevOps2022Fall&topic=DevOps-alerts-topics-lab09"

```

5. Learn more about Alertmanager:
<https://prometheus.io/docs/alerting/latest/alertmanager/>
6. Get the webhook URL following the steps here:
<https://zulip.com/integrations/doc/alertmanager> . Provide your full name in the Full Name field.

Update a `prometheus.yml` file with the path to `rules.yml`, and URL endpoint of alertmanager service (<k8s-controller_VM_EXT_IP>:9093).

Refer the following sample yml code block to update `prometheus.yml`.

Filename: `/monitor/prometheus.yml`

```

### Previous CONTENT

rule_files:
- rules.yml

alerting:
  alertmanagers:
    - follow_redirects: true
      scheme: http
      static_configs:
        - targets: ['<k8s-controller_VM_EXT_IP>:9093']

### Previous content

```

7. It's time to create an alertmanager container in the k8s-controller VM. For this you may need to update the `monitor/install_services.sh` file with the following docker command. Go through each setting and see if something should be updated.

```

docker run -d -p 9093:9093 \
  -v $PWD/monitor/alertmanager.yml:/alertmanager.yml \
  --name $ALERT_CONTAINER prom/alertmanager \
  --config.file=/alertmanager.yml \
  --cluster.advertise-address=<k8s-master_VM_EXT_IP>:9093

```

8. In a similar way, you also need to update the docker command where you were installing prometheus. Now you should bind mount the `rules.yml` file to the container's `/etc/prometheus/rules.yml` file. The final docker command may look like below:

```

docker run -d -p 9090:9090 \
  -v $PWD/monitor/rules.yml:/etc/prometheus/rules.yml \
  -v $PWD/monitor/prometheus.yml:/etc/prometheus/prometheus.yml \

```



```
--name $PROMO_CONTAINER prom/prometheus
```

9. You might need to comment all grafana related commands in the `monitor/install_services.sh` file.
10. Commit and push the changes with message `Deliverable: step 5.1, added alertmanager services.`

Step 5.2 Test the alert notification

1. Once the above mentioned job is succeeded, Goto prometheus service <http://k8s-controller VM EXT IP:9090> and click on the tab **Alerts** to find the alerts in *Inactive*, *Pending* and *Firing* state.
2. **Intentionally stop the flask application deployed on k8s-master VM to check if the alert configuration.**
3. Finally the alert signal should show in the Firing state as shown below. This means, Prometheus triggered alertmanager to push the notifications.
4. At the same time, you should be able to get the notification in Zulip topic.

FINAL NOTE

Remember that, we may see the commit history while grading your submission. Further, please zip your GitLab project and submit through the course wiki page.

Submission:

1. Download code of your GitLab project
2. Zip the code, screenshot and Upload the zip file to the course wiki page.
3. You may **Stop** the Virtual Machines and you can start using the same in the next **practice session**.

Don't delete your VMs

BONUS TASK

- In Step 2.3, you are manually adding prometheus as a datasource. Similarly, in the same step, you are manually creating the grafana container monitoring dashboard from the content of a json file. In Step 3.2, you are manually creating the grafana VM monitoring dashboard from the content of another json file.
- Make necessary changes (to **monitor** directory) so that both the dashboards and data sources will be added by Gitlab-CI.

- Create alerts for node-exporter, cAdvisor, and influxdb services.
 - An alert should be sent to the Zulip topic ([DevOps-alerts-topics-lab09](#)) when any service on a virtual machine is unavailable/stopped.